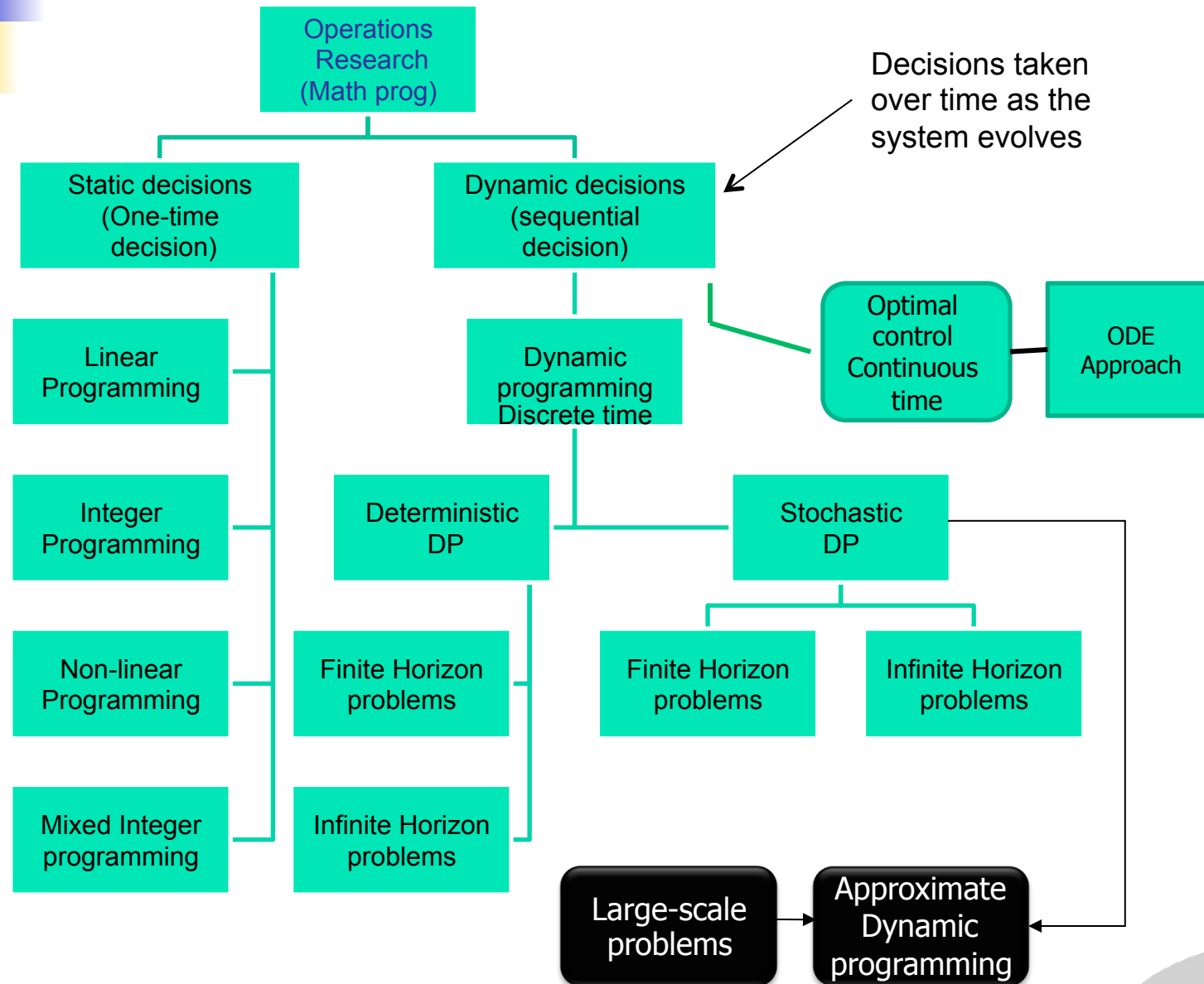


# Optimization in Prescriptive Analytics

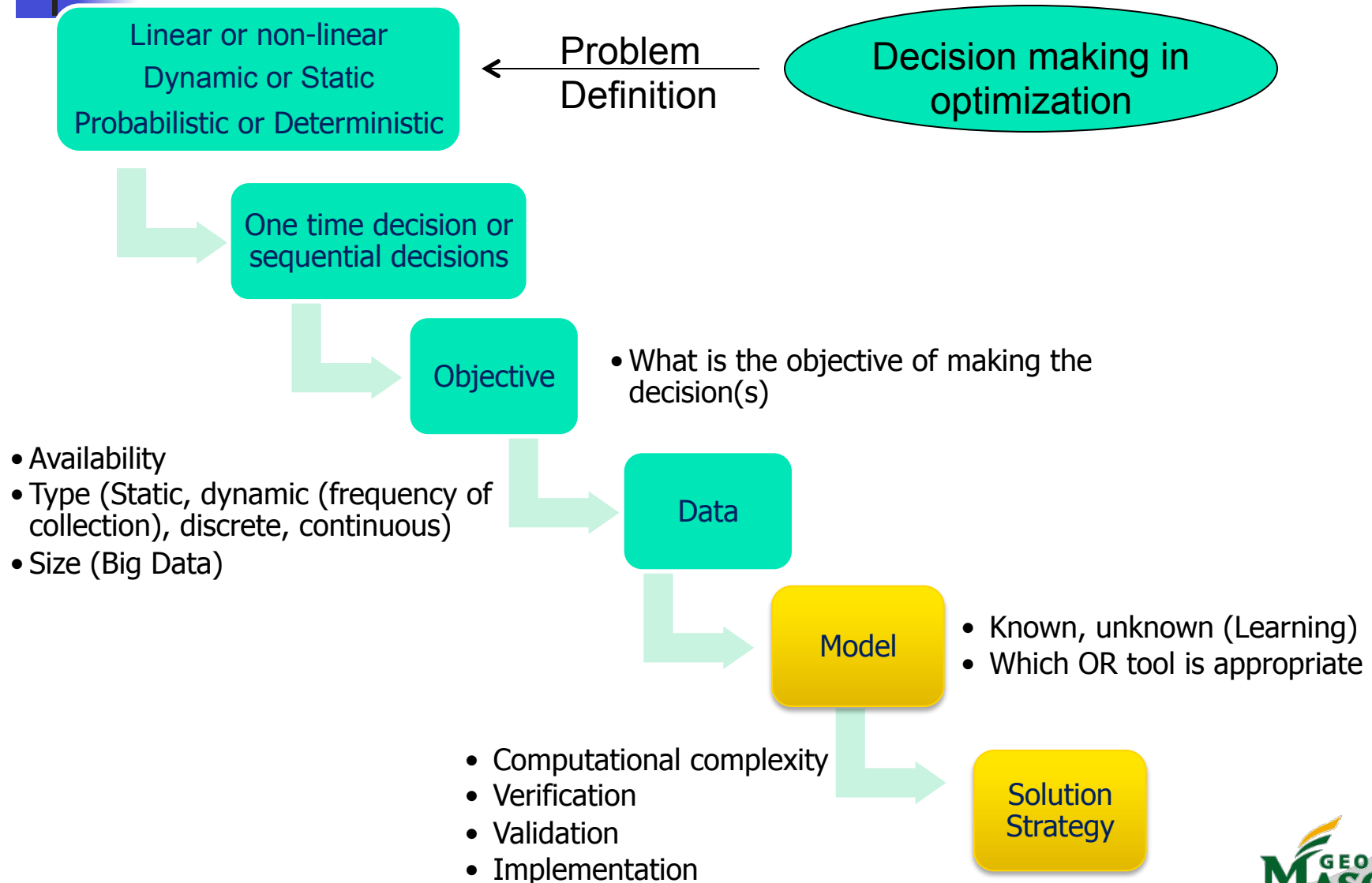
---

Rajesh Ganesan  
Associate Professor  
Systems Engineering and Operations Research  
George Mason University

# The Big Picture



# Thinking like an OR Analyst





# Static Decision Making

---

- Static (ignore time)
  - One time investment
  - Assignment
    - People to jobs
    - Jobs to machines (maximize throughput, minimize tardiness)
  - Min-cost Network flow (Oil pipelines)
  - Travelling Salesman (leave home, travel to different cities and return home in the shortest distance without revisiting a city)
  - Min Spanning Tree (Power/water lines)
  - Set Covering (Installation of fire stations)



# Dynamic Decision Making

---

- Dynamic (several decisions over time)
  - Portfolio management (monthly, yearly, or daily (day trader on Wall St))
  - Inventory control (hourly, daily, weekly ...)
  - Dynamic Assignment
    - Running a shuttle company (by the minute)
    - Fleet management for airline, trucks, locomotives
  - Airline seat pricing, Revenue management (by the hour)
  - Air traffic Control (by the minute)
  - Refinery process control (temperature control by the second)
  - Maneuvering a combat aircraft or a helicopter or a missile (decisions every millisecond)



# Today's Talk – A Few Select topics

## Modeling and Solution Strategies for Static and Dynamic Decision making

- Linear Programming example
  - Will tell you where it breaks down
- Integer Programming example
- What to do if the model is too hard to obtain or its simply not available and there is high computational complexity
  - Metaheuristics (directly search the solution space)
  - Simulation based Optimization
- Dynamic Programming example
- Computational aspects



# Linear Programming

- 100 workers
- 80 acres of land
- 1 acre of land produces 1 ton of either crop
- 2 workers are needed for every ton of either crop
- Your storage permits only a max production of 40 tons of wheat
- Selling price of wheat = \$3/ton
- Selling price of corn = \$2/ton

$x_1$  = quantity of wheat to grow in # of tons

$x_2$  = quantity of corn to grow in # of tons

## Mathematical Model

$$\text{Max } Z = 3x_1 + 2x_2$$

Subject to

$$2x_1 + 2x_2 \leq 100$$

$$x_1 + x_2 \leq 80$$

$$x_1 \leq 40$$

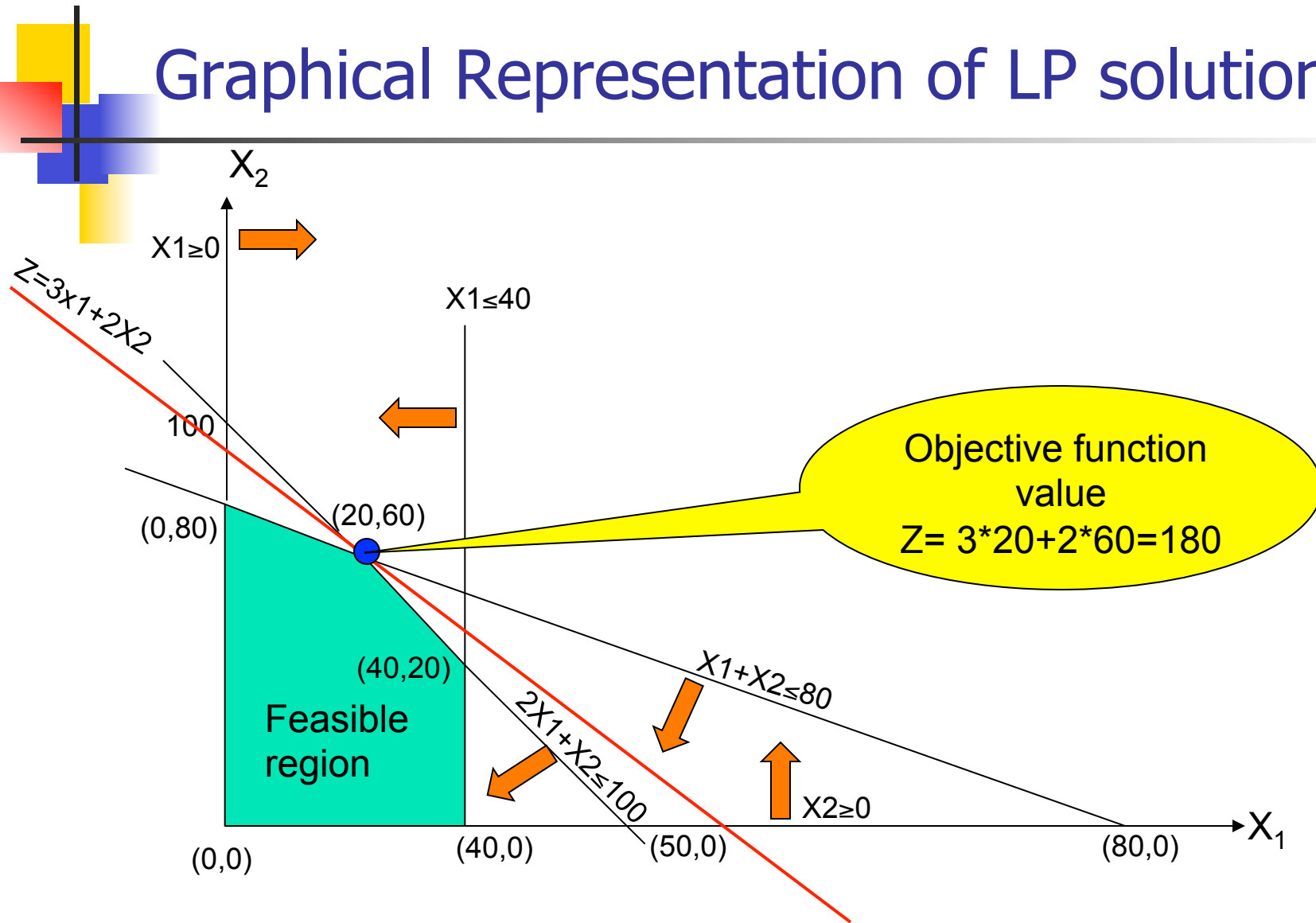
$$x_1 \geq 0$$

$$x_2 \geq 0$$

How many tons of wheat and corn to produce to maximize revenue?

Solution: Simplex Algorithm, solved using solvers, CPLEX, MLP software.

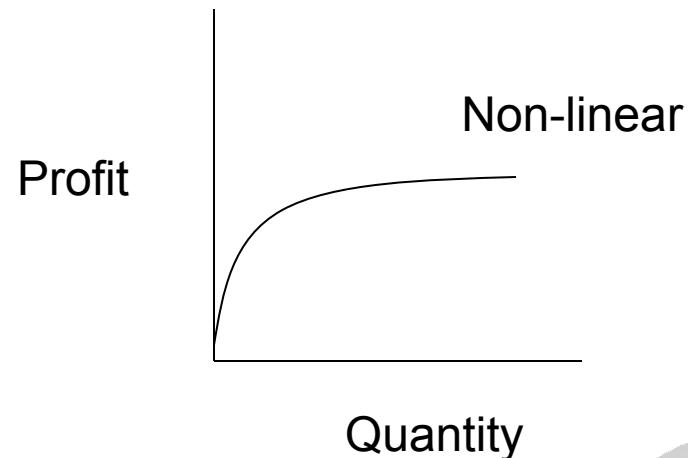
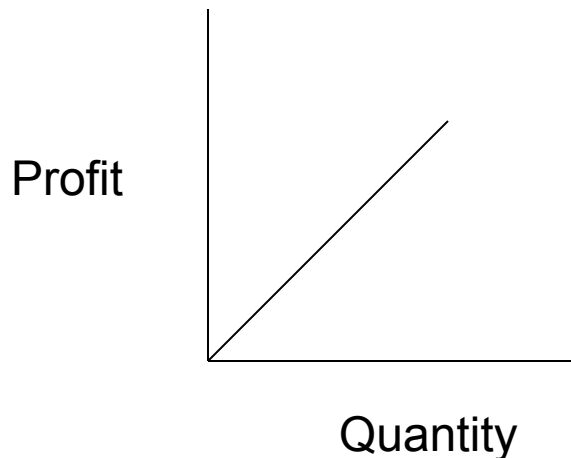
# Graphical Representation of LP solution





# Assumptions for LP to work

- 1. Proportionality: This is guaranteed if the objective and constraints are linear.
- 2. Additive: Independent decision variables
- 3. Divisibility: Fractions allowed
- 4. Certainty: Coefficients in the objective function and constraints must be fixed





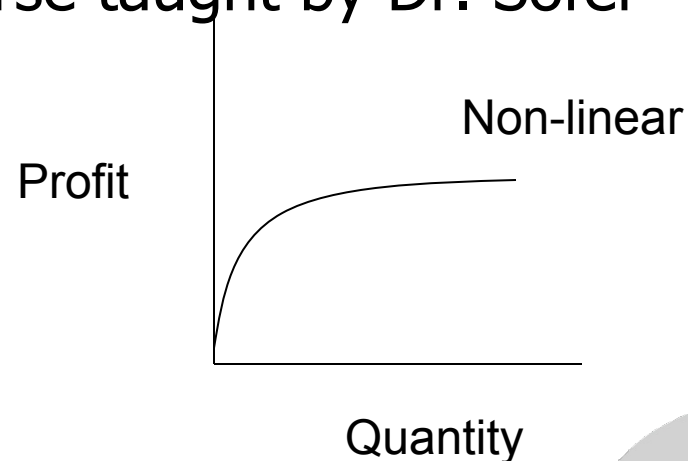
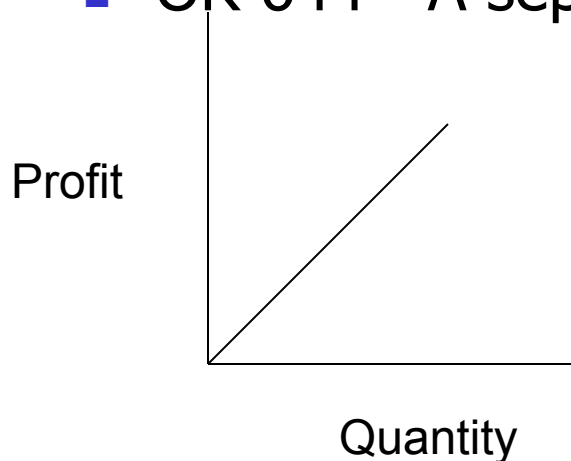
# What if you had many decision variables

---

- Big Data
  - Computational burden
    - Today's solvers can handle large problems
  - LP is easy to implement
    - Industry uses it a lot, often when they should not
  - Provides quick solutions
    - However, solutions can far from optimal if applied to problems under uncertainty in a non-linear environment.
    - So use caution. Use only when appropriate.
  - Is the real-world linear, fixed, deterministic?

# Relax Assumption 1

- 1. Proportionality: if not true
- $\text{Max } Z = 3x_1^2 + 2x_2^2$
- **Need Non-linear Programming (far more difficult than LP)**
- Solution strategies are very different
  - Method of steepest ascent, Lagrangian Multipliers, Kuhn-Tucker methods
- OR 644 - A separate course taught by Dr. Sofer





## Assumption 3: What if the decision variables are integers

---

- 3. Divisibility: If fractions are not allowed
- Yes or no decisions (0,1) binary variables
- Assignment problems
- **Need Integer Programming**
- OR 642 - A separate course taught by Dr. Hoffman
- These problems are more difficult to solve than LP



# Integer programming example

---

- Knapsack Problem (ever packed a suitcase?)
- Fill a sack with multiple units of items 1,2,& 3 such that the total weight does not exceed 10 lb and the benefit of the sack is maximum.

Item	benefit	wt
1	10	3
2	12	4
3	5	2



## Integer programming example

---

Item	benefit	wt
1	10	3
2	12	4
3	5	2

Let  $x_1$  be quantity of item 1. Similarly  $x_2$  and  $x_3$ . All  $x_1$ ,  $x_2$  and  $x_3$  are integers  $\geq 0$

$$\text{Max } 10 x_1 + 12 x_2 + 5 x_3$$

Subject to

$$3 x_1 + 4 x_2 + 2 x_3 \leq 10$$

$x_1, x_2, x_3$  are integers and  $\geq 0$

-Not a very easy problem to solve  
(you almost always forget something to  
pack that is important)

-This toy example- you can do it like  
solving a puzzle



# Computational complexity

---

- Now, try packing a UPS/FEDEX truck or aircraft with both weight & volume constraints and maximize the benefit
  - How many possible ways can you do this?
  - Although computers can help to solve, the solution is often not optimal.
    - **Computationally complex**
  - So we strive for near-optimal (good enough) solutions



# Computational complexity

---

- Traveling Salesman problem
  - Visit 20 cities and do not repeat a city and do all this by travelling the shortest distance overall
  - Distance between cities are known





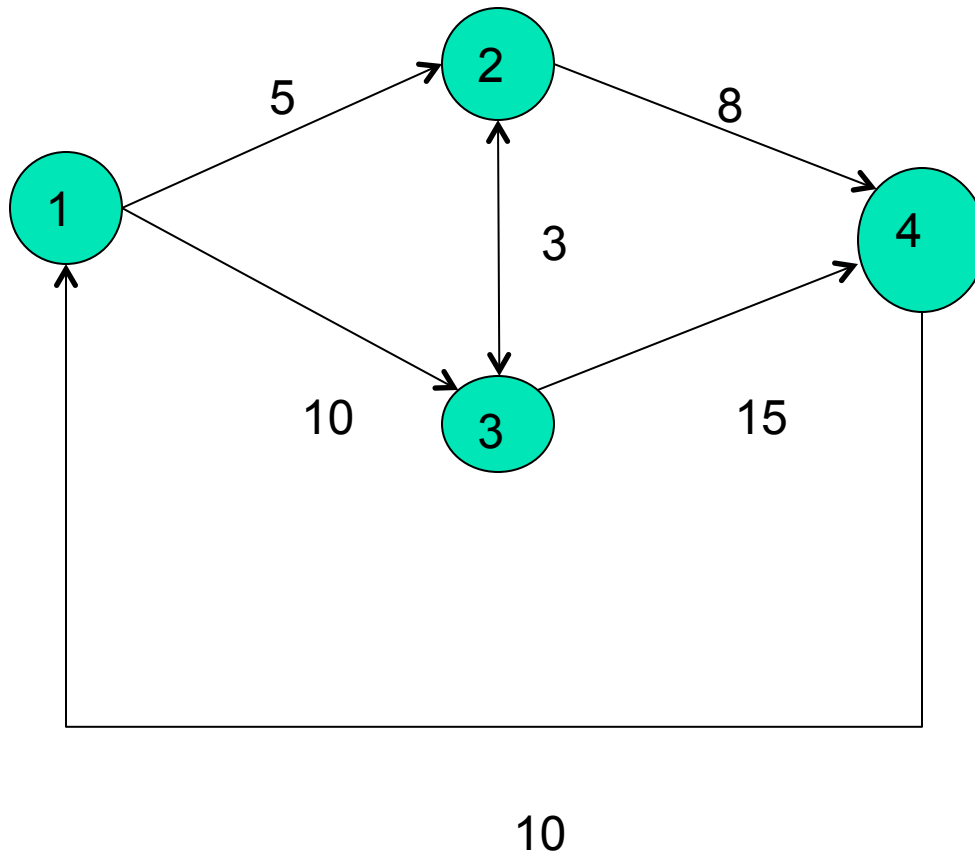
# Traveling Salesman Problem

---

- What if you chose the closest city from your current city every time?
- Only 20 solutions to evaluate (starting once from each city).
- Will you reach optimality?

# Perils of Short-term vision

4 city traveling salesman starting from city 1  
Must travel every city only once and return to 1



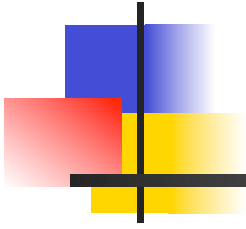
$$1-2-3-4-1 = 33$$

$$1-3-2-4-1 = 31$$

Myopic vision if  
You chose to go from  
City 1 to 2 because  
5 is smaller than 10

# Computational complexity

- Traveling Salesman problem
  - Visit 20 cities and do not repeat a city and do all this by travelling the shortest distance overall
  - Distance between cities are known
  - Today, there exists no computer to solve this “**optimally**” on Earth
  - X- - - - -X
  - $20 \times 19 \times 18 \times \dots \times 2 \times 1 = 20!$  Solutions =  $2 \times 10^{18}$  solutions
  - If a computer can evaluate 100 million solutions per second, it will take 771 years!!!



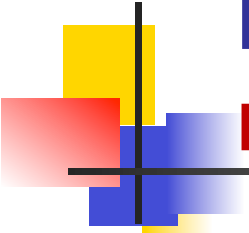
So we strive of near-optimal (good enough) solutions

In

Big data problems

With

Severe computational complexity



# Metaheuristics is one way to solve TSP near-optimally

---

- Several techniques
- Search the solution space
- **There are no models like LP, IP, NLP**
- Start your search by defining one or many feasible solutions
- Improve your objective of the search by tweaking your solutions systematically
- Stop search when you have had enough of it (computing time reaches your tolerance)
- Be happy with the solution that you have at that point
- You may have gotten the optimal solution but you will never know that it is indeed optimal



# Reaching real-world conditions

---

- Let us introduce dynamic decision over time and uncertainty

on top of

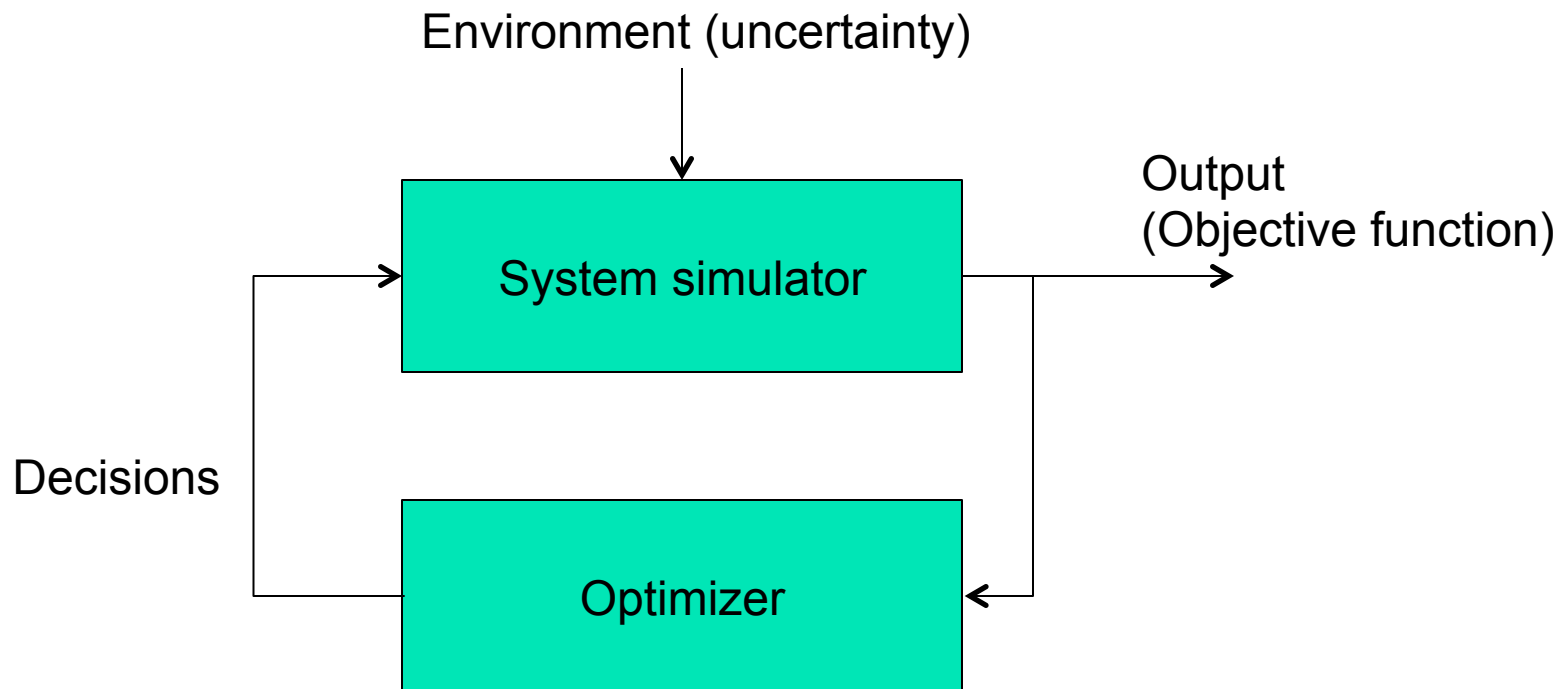
Big data

Complex non-linear system

Computational difficulty

# Need model-free approaches

## ■ Simulation-based Optimization



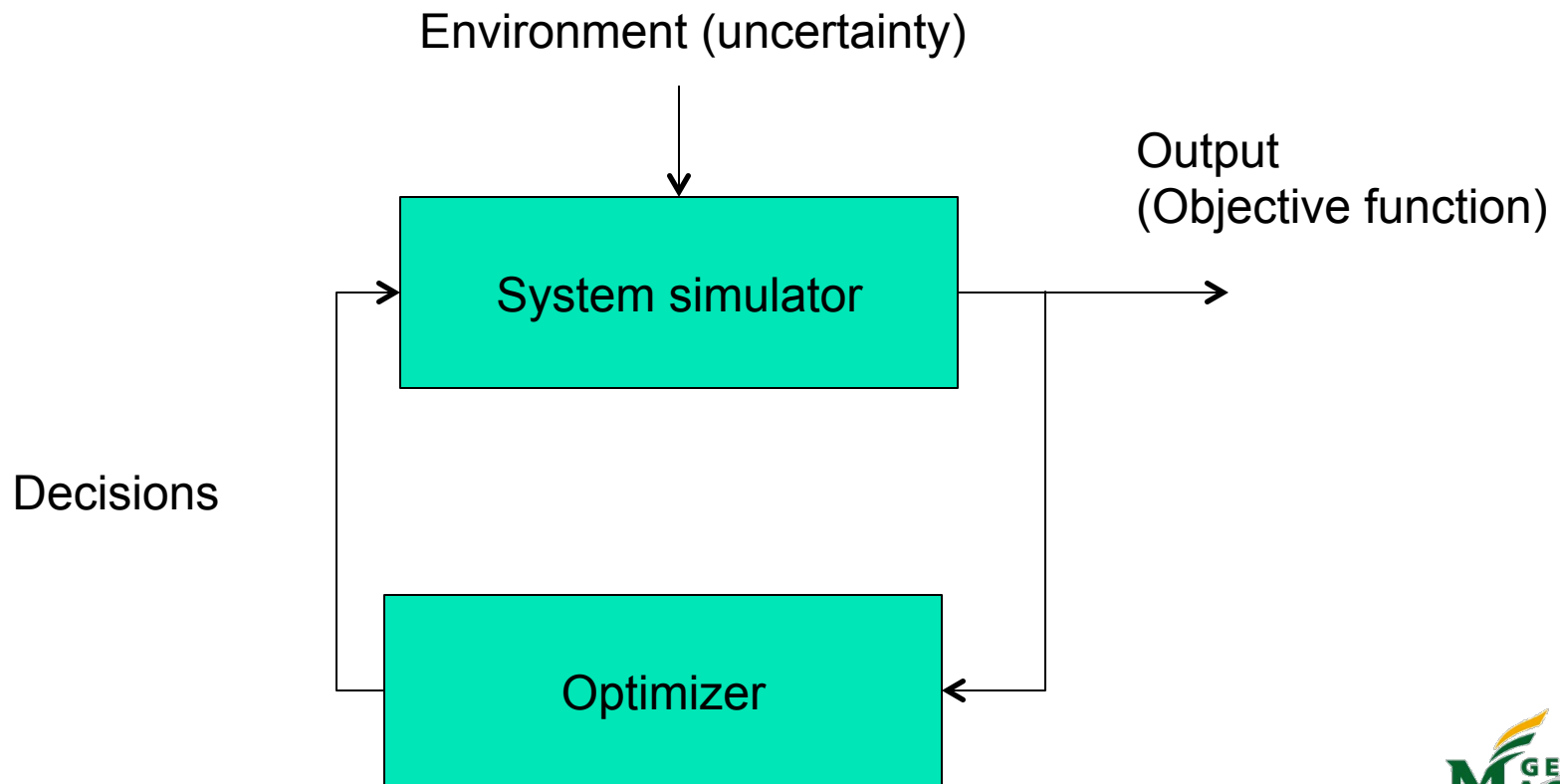
Simple example: Car on cruise control

A mathematical model that relates all car parameters and the environment parameters may not exist

A more difficult to solve and complex example: Air traffic control

# Simulation-based Optimization

- In more complex problems such as helicopter control the Optimizer is an Artificial Intelligence (Learning) Agent







# How does an AI agent learn?

---

- In a discrete setting you need **Dynamic Programming** (OR674 and OR 774) Join my class!! – term common among advanced OR
- In a continuous time setting it is called optimal control (Differential equations are used) – term common among Electrical Engineers
- Mathematically the above methods are IDENTICAL
- Computer science folks call it machine learning, AI, or Reinforcement learning and use it for computer games

Learning happens in two ways: supervised and unsupervised



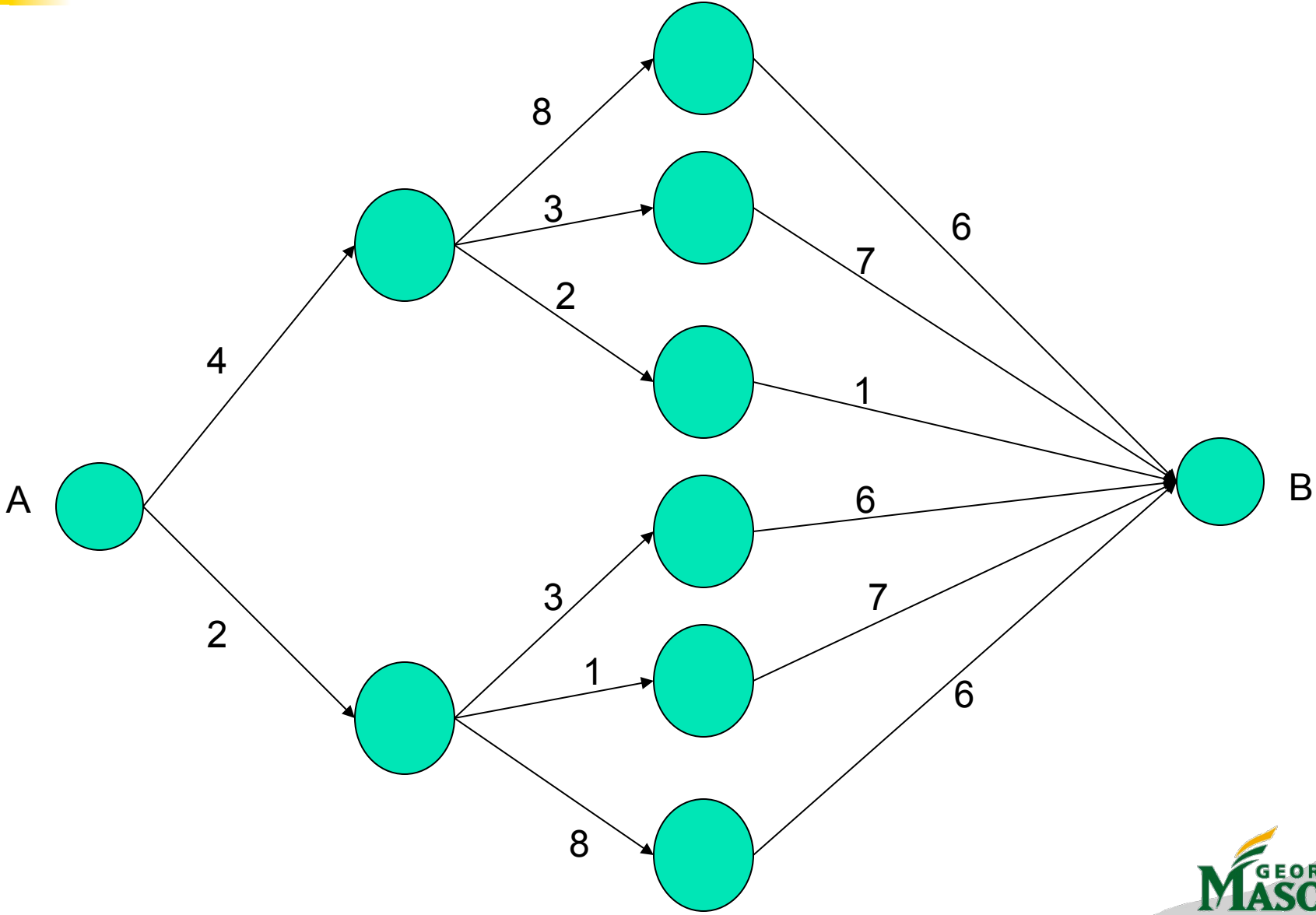
# Dynamic programming

---

What is it?

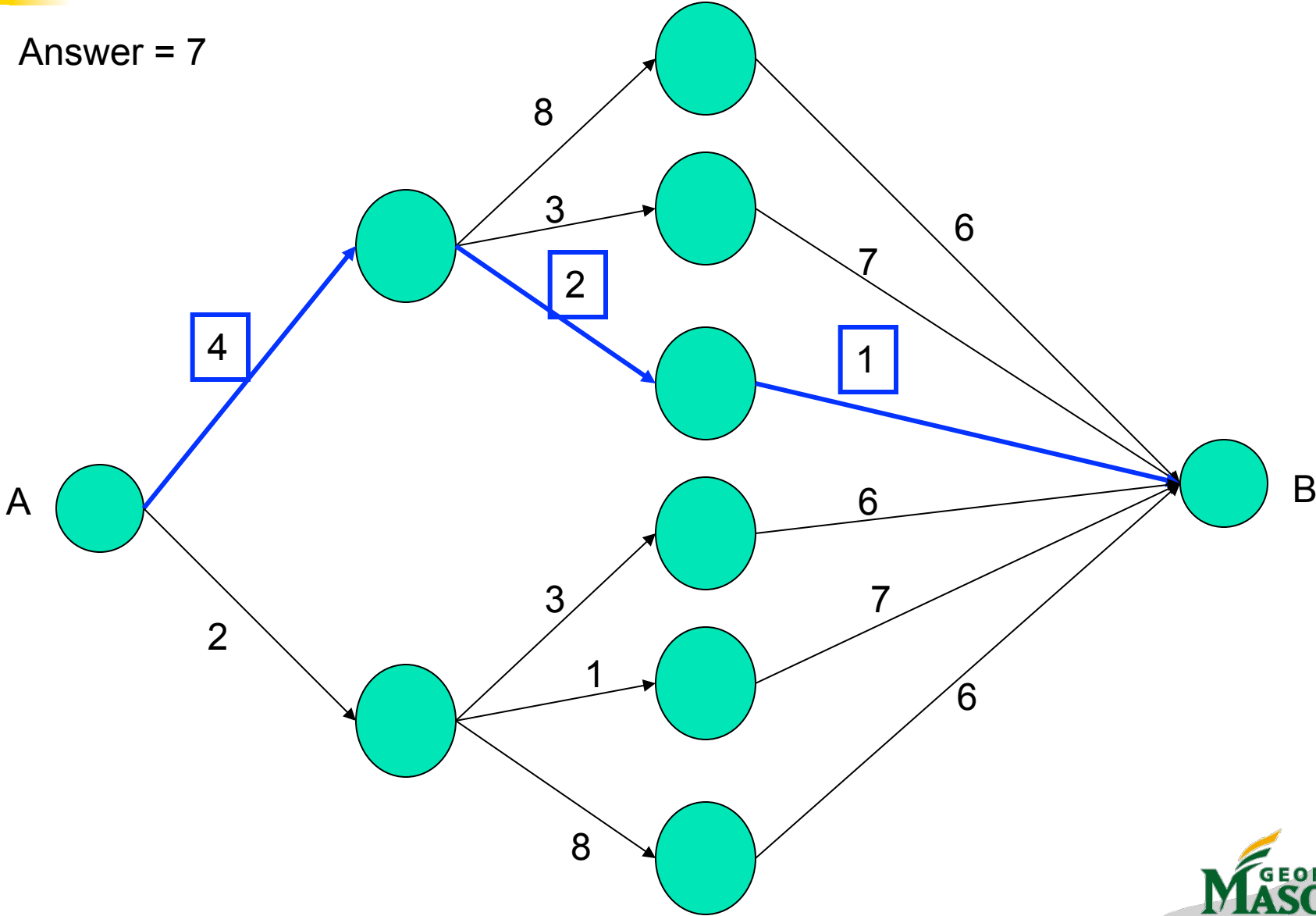
Operates by finding the shortest path (minimize cost) or longest path (maximize reward) of decision making problems that are solved over time

Find the shortest path from A to B



# Find the shortest path from A to B

Answer = 7





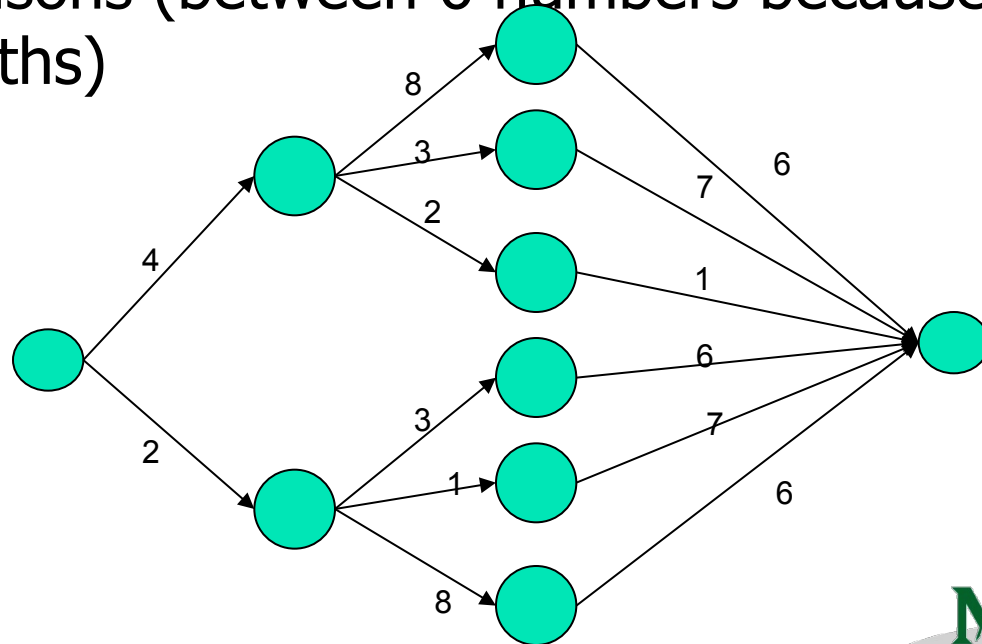
# Questions

---

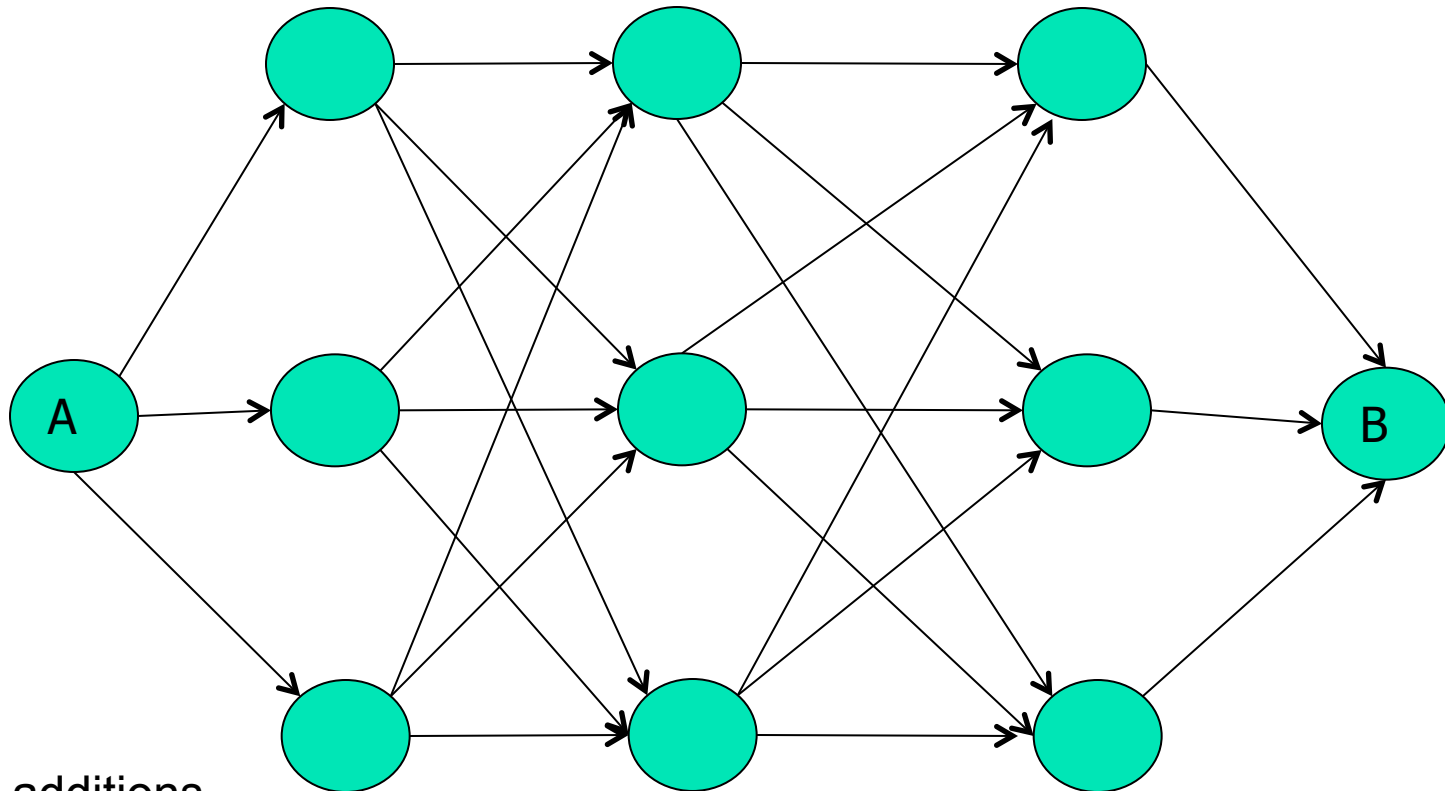
- How many of you evaluated all possible paths to arrive at the answer?
  - How many of you started by looking at the smallest number from A (in this case it is 2) and went on to the next node to find the next smallest number 1 to add and then added 7 to get an answer of 10
  - If you did all possible paths then you performed an explicit enumeration of all possible paths (you will need 771 years or more to solve 20 city TSP)
- or
- you tried to follow a myopic (short-sight) policy, which did not give the correct answer

# From a Computer's stand point

- For explicit enumeration, to find the shortest path
  - There were 18 additions
  - And 5 comparisons (between 6 numbers because there are 6 paths)



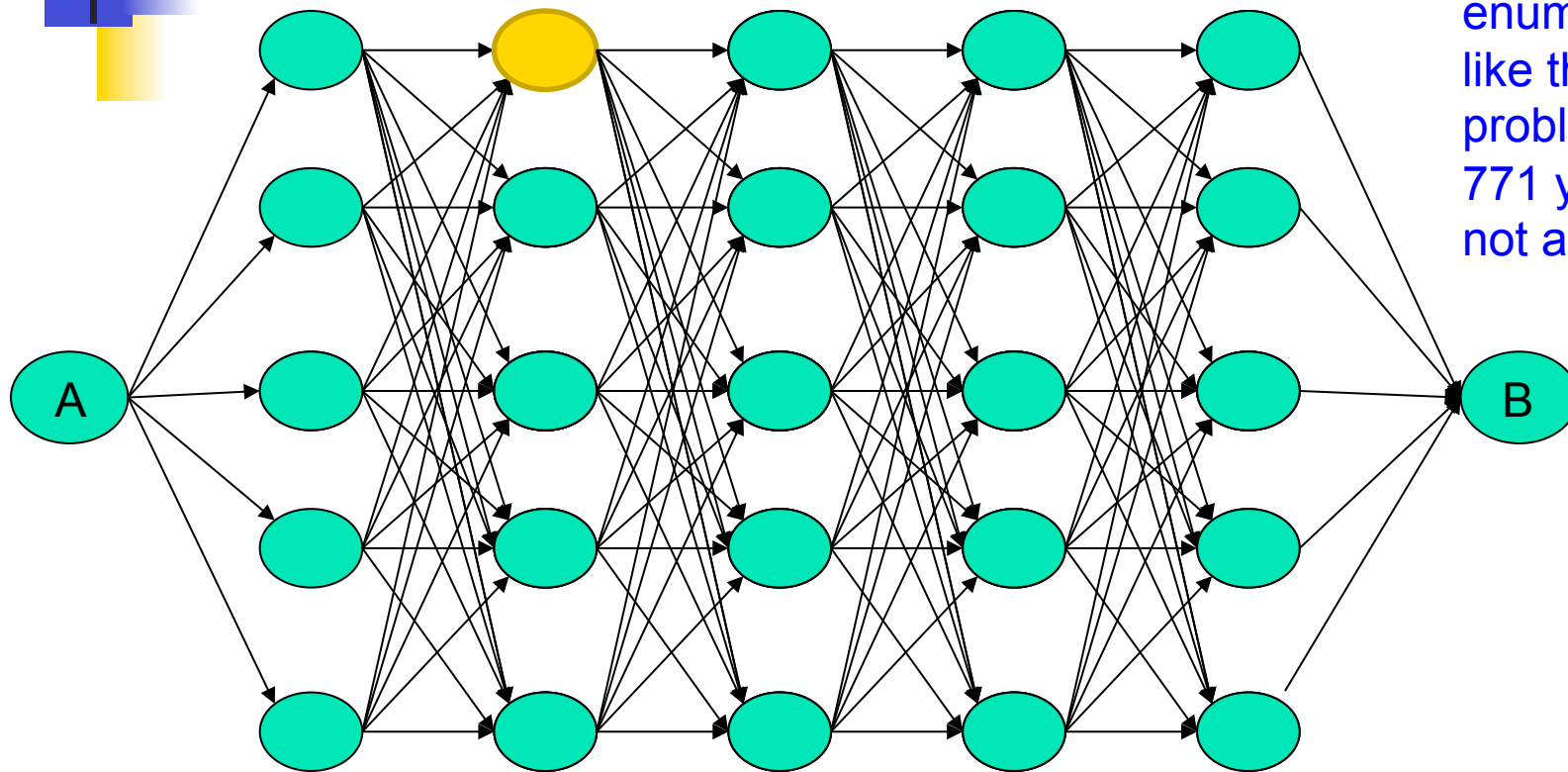
# Another example



27 paths  
 $27 \cdot 3 = 81$  additions  
26 comparisons

## Another example

Exhaustive enumeration like the TSP problem with 771 years is not an option)



Explicit enumeration

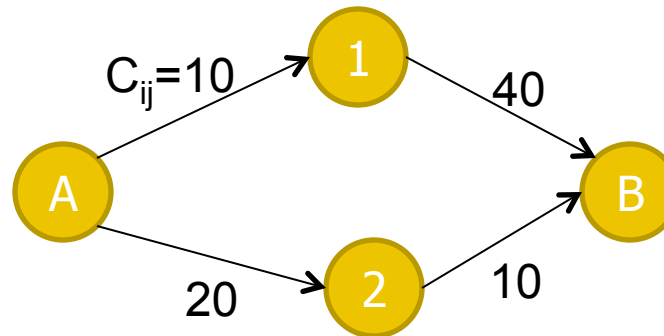
$5^5$  paths \* 5 additions per path = 15625 additions

$5^5 - 1$  comparisons = 3124



# Myopic vs DP thinking - find shortest path from A to B

$C_{ij}$  = cost on the arc



Myopic policy:  $V(A) = \min(C_{ij})$   
= min of (10 or 20)  
leads to solution of 50 from A to 1 to B

DP policy:  $V(A) = \min(C_{ij} + V(\text{next node}))$   
= min (10 + 40, 20 + 10) = 30  
leads to solution of 30 from A to 2 to B

Key is to find the values of node 1 and 2  
How? By learning via simulation-based optimization

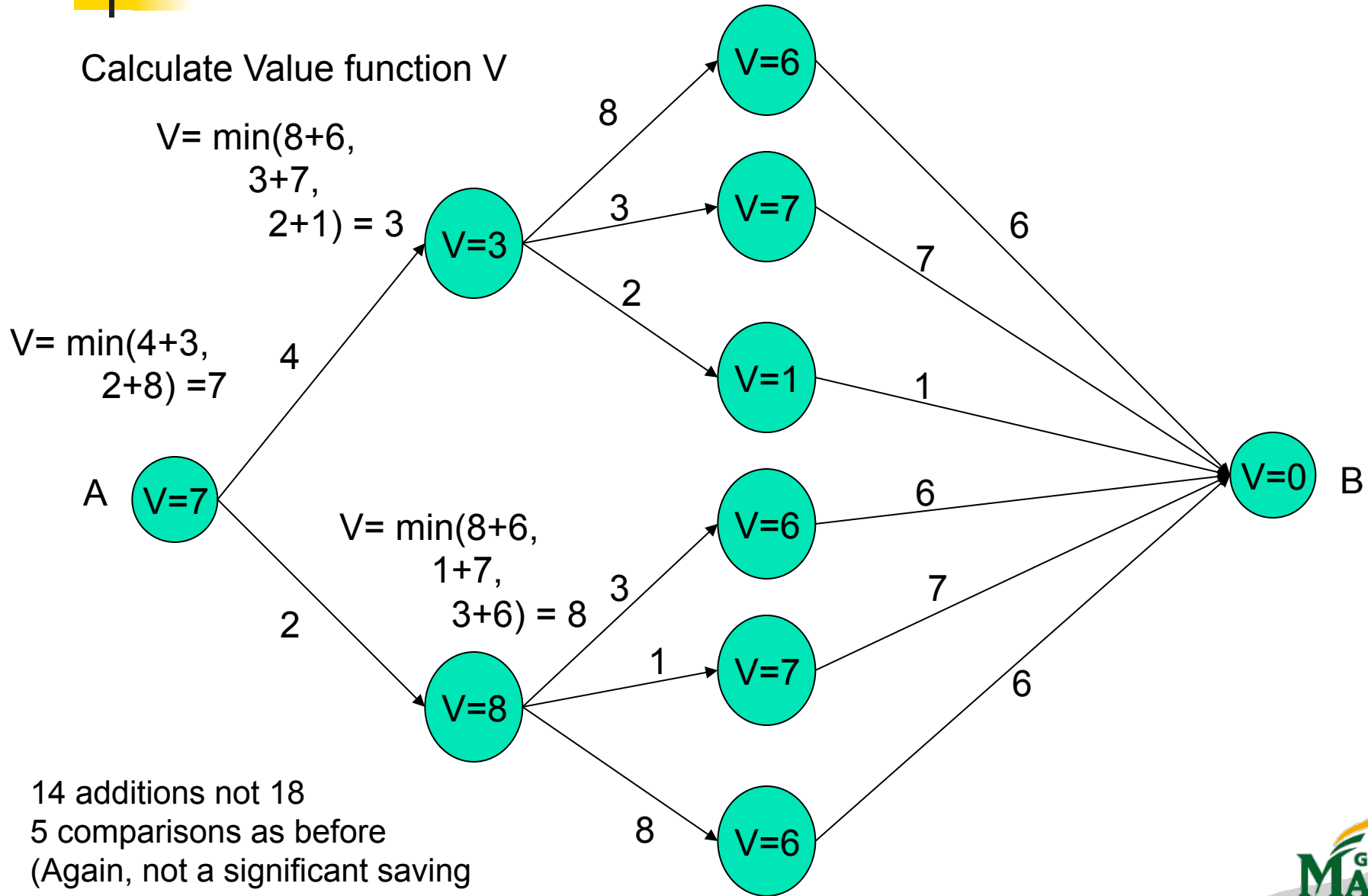
# Backward recursion

Calculate Value function V

$$V = \min(8+6, 3+7, 2+1) = 3$$

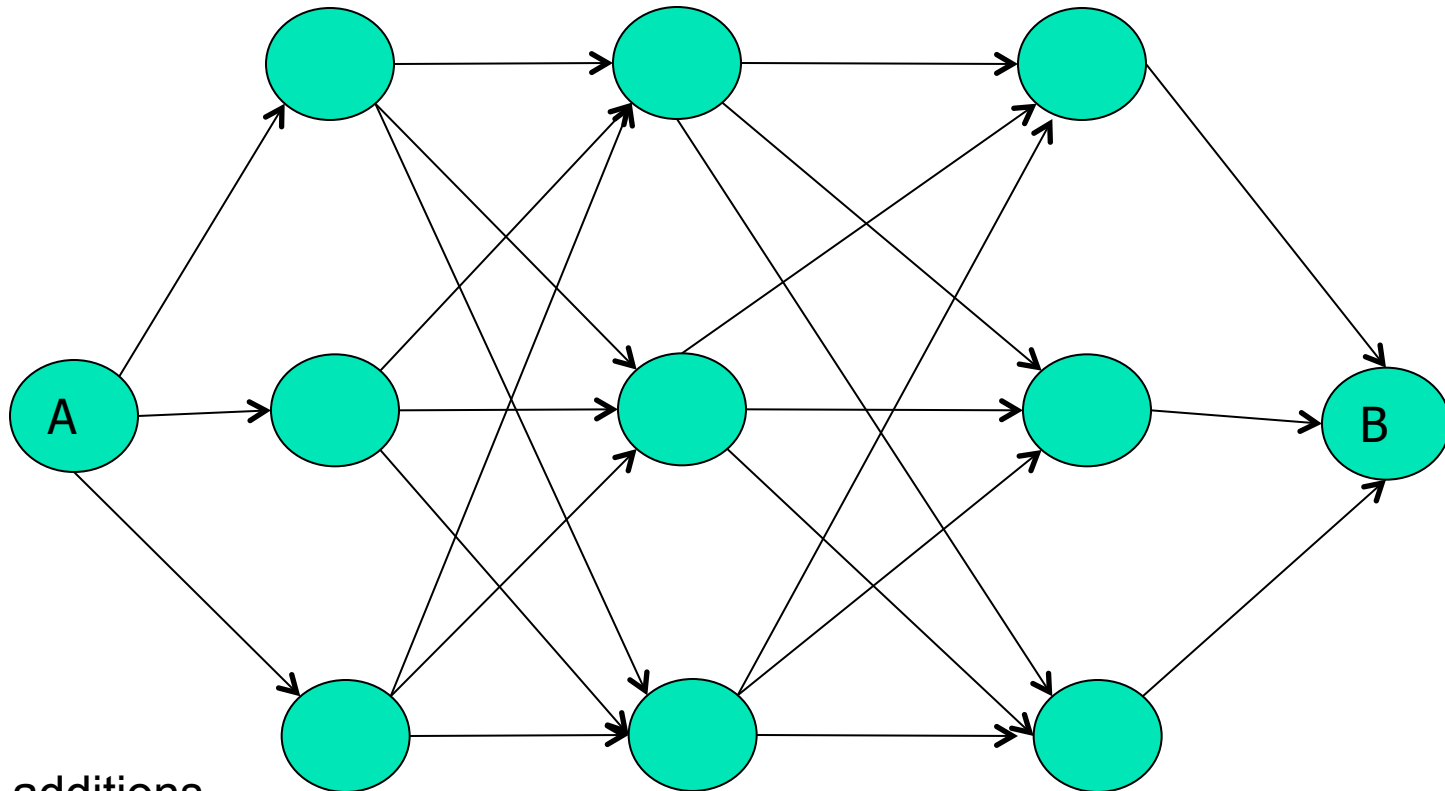
$$V = \min(4+3, 2+8) = 7$$

$$V = \min(8+6, 1+7, 3+6) = 8$$



14 additions not 18  
 5 comparisons as before  
 (Again, not a significant saving  
 in computation)

# Another example

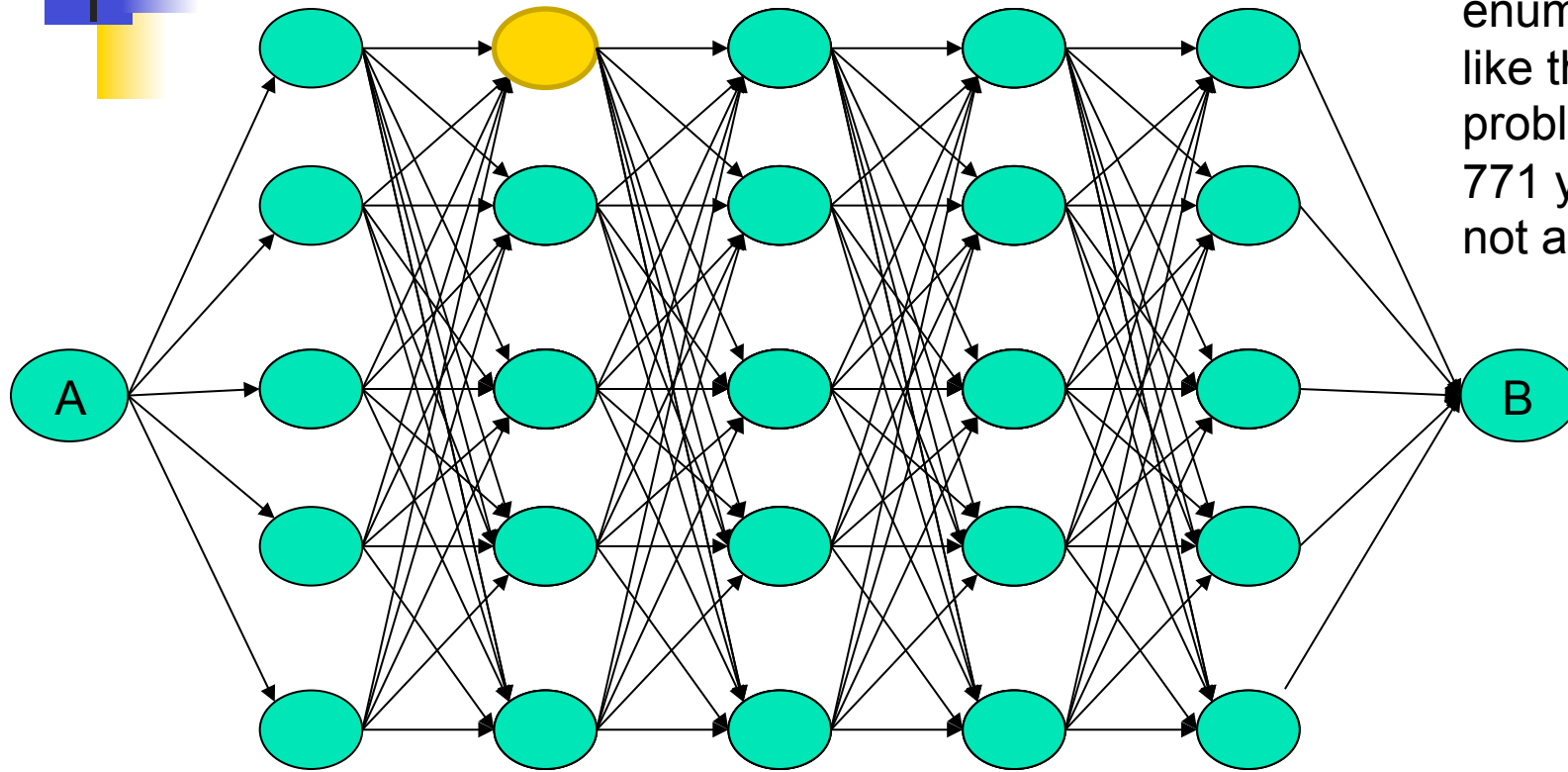


27 paths  
 $27 \cdot 3 = 81$  additions  
26 comparisons

Backward recursion  
24 additions  
13 comparisons

## Another example

Exhaustive enumeration like the TSP problem with 771 years is not an option)



With backward recursion  
 $4 \cdot (25) + 10 = 110$  additions  
 $20 \cdot 4 + 1$  comparisons = 81

Explicit enumeration

$5^5$  paths \* 5 additions per path = 15625 additions

$5^5 - 1$  comparisons = 3124

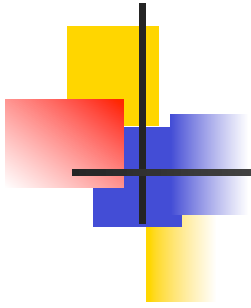
**Wow.. That is a significant saving in computation!!!!**



## Backward Recursion

---

- Real world problems cannot be solved backwards because time flows forward
- So we need to estimate the **value** of the future **states**
- We estimate the **value** of the future **states almost accurately** by **unsupervised learning in a simulator which interacts with the environment.**
- We make random decisions initially and learn from those and then become greedy eventually by making only the best decisions.



Welcome to the field of  
**Dynamic Programming!!**  
for  
**Sequential Decision Making (over time)**  
based on  
the idea that

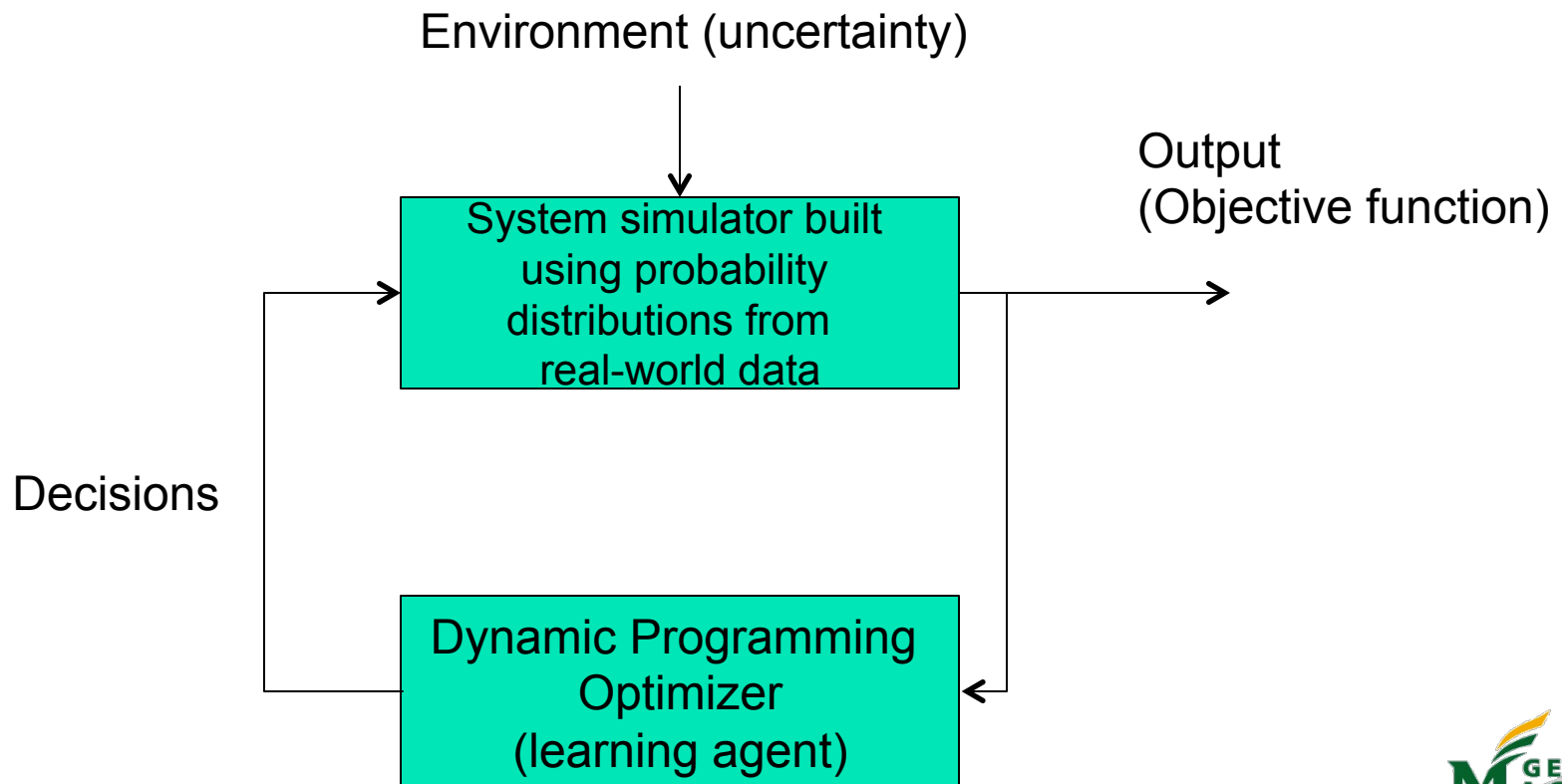
We want to move from one good state of the system to another  
by  
making an near-optimal decision  
in the presence of uncertainty

It also means that the future state depends only on the current and the decision taken  
in the presence of uncertainty (and not on the past – memory-less property)

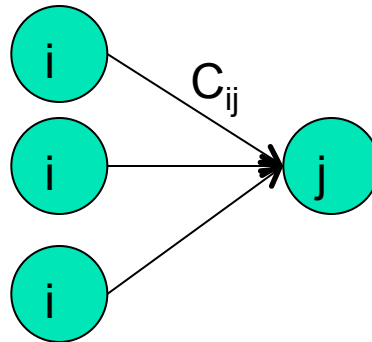
The above is achieved via unsupervised learning that entails only  
an interaction with the environment in a model-free setting

# Simulation-based Optimization

- In more complex problems such as helicopter control the Optimizer is an Artificial Intelligence (Learning) Agent



# Formal Definition



Formally, Backward Recursion  
$$V_t(i) = \max \text{ or } \min [C_{ij} + V_{t+1}(j)] , i < j$$
$$j$$

The above equation is modified to include uncertainty  
The theory of Markov decision process also called stochastic dynamic Programming

Key is to find the values of node j while solving V(node i)  
How? By learning via simulation-based optimization





# Examples of DP in the real world

---

- Problems with uncertainty, big data, computationally difficult (not widespread as LP)
  - MapQuest, Google maps
  - HVAC control
  - Helicopter control
  - Missile control
  - Schneider National (trucking)
  - Blood bank Inventory control
  - Financial Economics
  - Competitive Games (game theory)
    - Based on Nash equilibrium – (Movie: Beautiful mind acted by Russell Crowe)
    - Power distribution in the US North East Corridor
    - Revenue management (Airline seat pricing)



# Computational Aspects

---

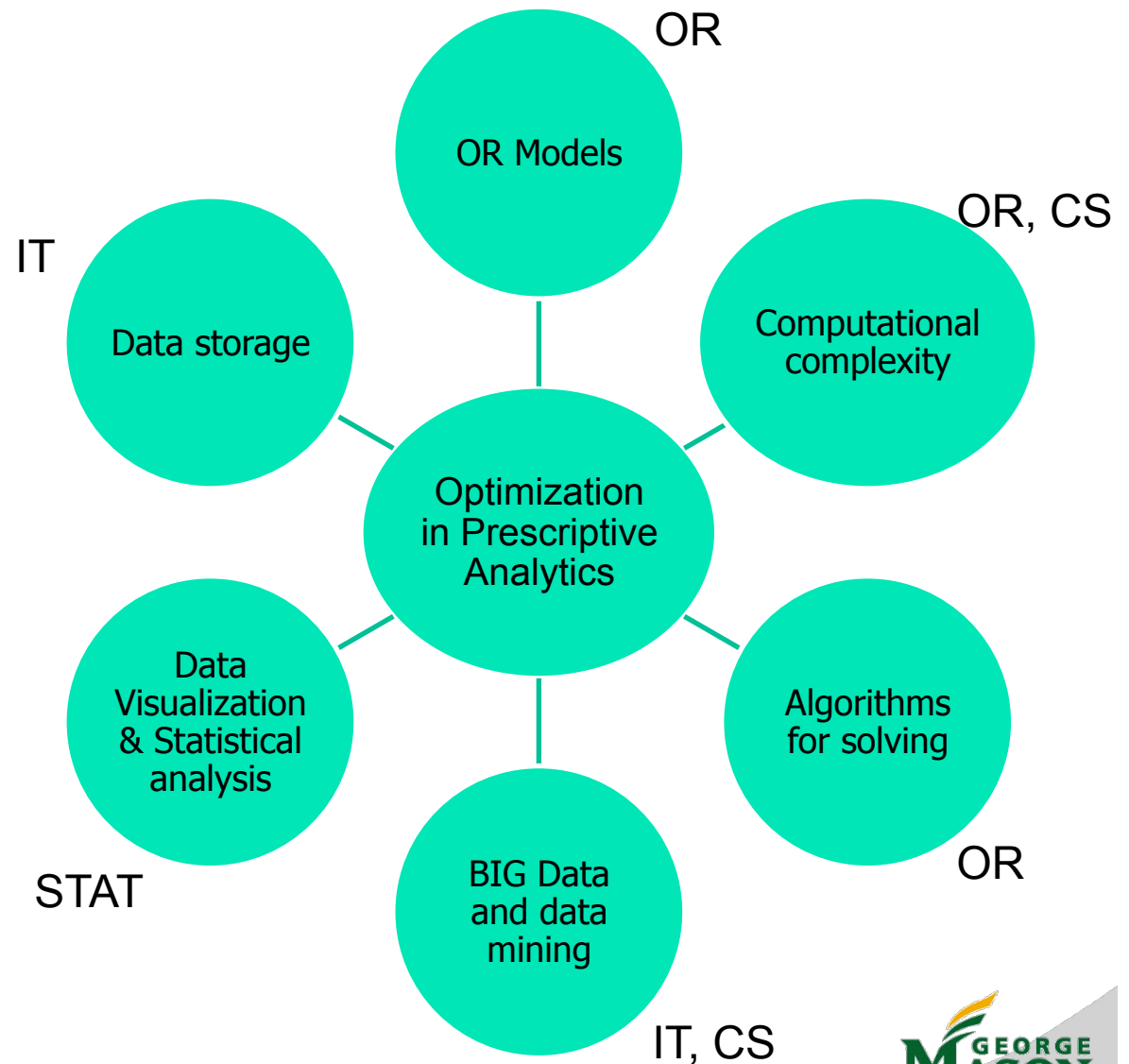
- LP - software has been developed. It has been widely researched
- IP and NLP are more difficult to solve than LP (software exists). Well researched
- DP is not well researched and is a newer field (no softwares, have to write the code)
  - Computationally far difficult than LP, IP, NLP but we are getting better with faster computers
  - However, DP is the only route for near-optimally solving some of the toughest DYNAMIC optimization problems in the world
    - Particularly for sequential decision making every few seconds in a fast changing and uncertain environment
  - If you solve it, PATENT IT!!

# In Summary

The presentation is an overview (breadth) of select optimization methods capable of handling Big data, rather than the specifics (depth) of how to implement these methods for Big data Analytics

The talk was to create an awareness for the methods that are out there but not used widely in the real-world

Embracing them could make a lot of difference on how decisions are done for Big data driven complex systems operating under uncertainty





# In Summary – Main take away

---

## In Big Data decision making Problems (Prescriptive Analytics)

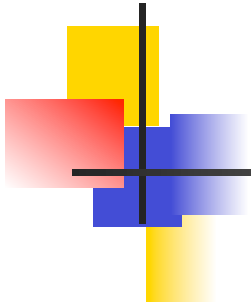
Understand characteristics of the data, linear/non-linear, deterministic or probabilistic, static or dynamic (frequency of collection)

Beware of

1. Myopic policies
2. Exhaustive enumeration of all solutions
3. Computational Complexity

Look for appropriate modeling and solution strategies that can provide near-optimal decisions (good-enough) In the long run for the problem at hand.

When making a decision sequentially over time, make sure to sum the cost/reward of making the decision with the value of the estimated future state that the decision will take you to. Then pick a decision that minimizes (if cost) or maximizes (if reward) the above sum.



Thank You